1. Load The Dataset

In [1]:

```
#Import all the necessary libraries and print their versions
import sys
import nltk
import sklearn
import pandas
import numpy

print('Python: {}'.format(sys.version))
print('NLTK: {}'.format(nltk.__version__))
print('Scikit-learn: {}'.format(sklearn.__version__))
print('Pandas: {}'.format(pandas.__version__))
print('Numpy: {}'.format(numpy.__version__))
```

```
Python: 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
NLTK: 3.4.5
Scikit-learn: 0.21.3
Pandas: 0.25.1
Numpy: 1.16.5
```

In [2]:

```
# Load the Dataset
import pandas as pd
import numpy as np

data_source = "All_Review.xlsx"
df = pd.read_excel(data_source)
```

In [3]:

```
#Print important information of the dataset
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110 entries, 0 to 109
Data columns (total 5 columns):
Email                   110 non-null object
Restaurant              110 non-null object
Real_or_Fake            110 non-null object
Positive_or_Negative    110 non-null object
Reviews                 110 non-null object
dtypes: object(5)
memory usage: 4.4+ KB
None
                     Email Restaurant Real_or_Fake Positive_or_Negative  \
0  sagar.lamichhane@ttu.edu          T            F                     P
1  sagar.lamichhane@ttu.edu          T            F                     N
2  sagar.lamichhane@ttu.edu          T            F                     N
3  sagar.lamichhane@ttu.edu          T            F                     N
4  sagar.lamichhane@ttu.edu          T            R                     P

                                             Reviews
0  The place is really great. Owner is from my vi...
1  For me, it was the worst place to have an expe...
2  It was a nightmare to go there and spend my ti...
3  I wonder why such place still exists. Why woul...
4  The food is very good and a break from the usu...
```

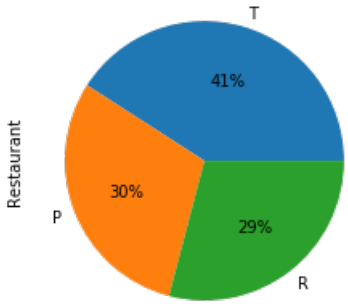
In [4]:

```
#view the informations of datasets in pie and bar plots
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df.Restaurant.value_counts().plot(kind='pie', autopct='%1.0f%%')
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c0e5157548>
```

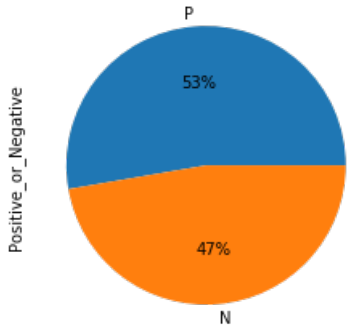


In [5]:

```
df.Positive_or_Negative.value_counts().plot(kind='pie', autopct='%1.0f%%')
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c0e547e308>
```

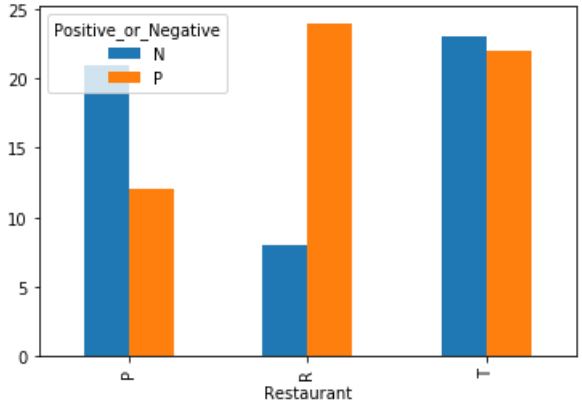


In [6]:

```
Positive_or_Negative = df.groupby(['Restaurant','Positive_or_Negative']).Positive_or_Negative.coun
t().unstack()
Positive_or_Negative.plot(kind='bar')
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c0e54e6548>
```

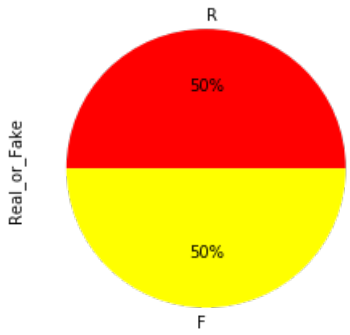


In [7]:

```
df.Real_or_Fake.value_counts().plot(kind='pie',autopct='%1.0f%%',colors=["red","yellow"])
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c0e5584588>
```



In [8]:

```
#check class distribution
classes = df.Real_or_Fake
print(classes.value_counts())
```

```
R    55
F    55
Name: Real_or_Fake, dtype: int64
```

2. Preprocess the Data

In [9]:

```
#convert class labels to binary values, 0 = Fake, 1 = Real

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
Y = encoder.fit_transform(classes)

print(classes[:10])
print(Y[:10])
```

```
0    F
1    F
2    F
3    F
4    R
5    R
6    R
7    R
8    F
9    F
Name: Real_or_Fake, dtype: object
[0 0 0 0 1 1 1 1 0 0]
```

In [10]:

```
#store the  Review datas
the_reviews = df.Reviews
print(the_reviews[:10])
```

```
0    The place is really great. Owner is from my vi...
1    For me, it was the worst place to have an expe...
2    It was a nightmare to go there and spend my ti...
3    I wonder why such place still exists. Why woul...
4    The food is very good and a break from the usu...
5    Came from out of town. Ordered from here. Chic...
```

```
6    If you are looking for an authentic Indian foo...
7    Just....Meh. We ordered samosas, southern curr...
8    Royal Indian is really a good place to have a ...
9    I really loved the food and the services they ...
Name: Reviews, dtype: object
```

In [11]:

```
# use regular expressions to replace email addresses, URLs, phone numbers, other numbers

# Replace email addresses with 'email'
processed = the_reviews.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                    'emailaddress')

# Replace URLs with 'webaddress'
processed = processed.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                    'webaddress')

# Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
processed = processed.str.replace(r'£|\$', 'moneysymb')

# Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'ph
onenumber'
processed = processed.str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                    'phonenumbr')

# Replace numbers with 'numbr'
processed = processed.str.replace(r'\d+(\.\d+)?', 'numbr')
```

In [12]:

```
# Remove punctuation
processed = processed.str.replace(r'[^\w\d\s]', ' ')

# Replace whitespace between terms with a single space
processed = processed.str.replace(r'\s+', ' ')

# Remove leading and trailing whitespace
processed = processed.str.replace(r'^\s+|\s+?$', '')
```

In [13]:

```
# change words to lower case
processed = processed.str.lower()
```

In [14]:

```
# remove stop words from text messages
from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

processed = processed.apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))
```

In [15]:

```
# Remove word stems using a Porter stemmer (ing, tenses, etc.)
ps = nltk.PorterStemmer()

processed = processed.apply(lambda x: ' '.join(
    ps.stem(term) for term in x.split()))
```

In [16]:

```
print (processed)
```

```
0      place realli great owner villag realli friendl...
1      worst place experi indian food food realli nas...
2      nightmar go spend time liter spend time money ...
```

```
3      wonder place still exist would peopl give good...
4      food good break usual daili find chicken birya...
                           ...
105    horribl experi wait long wonder custom furnitu...
106    went place yesterday see review know gave good...
107    misl india palac titl tri india palac outlet a...
108    owner hospit hardwork enjoy food weak spice me...
109    worst indian food tast buffet togeth famili nu...
Name: Reviews, Length: 110, dtype: object
```

3. Generating Features

In [17]:

```
# check the no. of most repeated words
from nltk.tokenize import word_tokenize

# create bag-of-words
all_words = []

for message in processed:
    words = word_tokenize(message)
    for w in words:
        all_words.append(w)

all_words = nltk.FreqDist(all_words)
```

In [18]:

```
# print the total number of words and the 15 most common words
print('Number of words: {}'.format(len(all_words)))
print('Most common words: {}'.format(all_words.most_common(15)))
```

```
Number of words: 1038
Most common words: [('food', 173), ('indian', 111), ('place', 80), ('restaur', 75), ('good', 55),
('numbr', 48), ('like', 46), ('realli', 45), ('chicken', 40), ('go', 37), ('order', 37), ('would',
36), ('time', 36), ('dish', 35), ('tri', 33)]
```

In [19]:

```
# use the 100 most common words as features
word_features = list(all_words.keys())[:100]
```

In [20]:

```
# The find_features function will determine which of the 100 word features are contained in the re
view
def find_features(message):
    words = word_tokenize(message)
    features = {}
    for word in word_features:
        features[word] = (word in words)

    return features

# Lets see an example on first index
features = find_features(processed[0])
for key, value in features.items():
    if value == True:
        print (key)
```

```
place
realli
great
owner
villag
friendli
come
talk
would
```

```
suggest
food
```

In [21]:

```
#printed om sentences
processed[0]
```

Out[21]:

```
'place realli great owner villag realli friendli come talk would suggest food'
```

In [22]:

```
#check if its in sentences or not
features
```

Out[22]:

```
{'place': True,
 'realli': True,
 'great': True,
 'owner': True,
 'villag': True,
 'friendli': True,
 'come': True,
 'talk': True,
 'would': True,
 'suggest': True,
 'food': True,
 'worst': False,
 'experi': False,
 'indian': False,
 'nasti': False,
 'menu': False,
 'provid': False,
 'look': False,
 'messi': False,
 'confus': False,
 'enjoy': False,
 'moment': False,
 'go': False,
 'tell': False,
 'friend': False,
 'famili': False,
 'visit': False,
 'ever': False,
 'life': False,
 'nightmar': False,
 'spend': False,
 'time': False,
 'liter': False,
 'money': False,
 'noth': False,
 'peopl': False,
 'know': False,
 'ho': False,
 'take': False,
 'order': False,
 'staff': False,
 'busi': False,
 'use': False,
 'phone': False,
 'unless': False,
 'call': False,
 'respond': False,
 'restaur': False,
 'wonder': False,
 'still': False,
 'exist': False,
 'give': False,
 'good': False,
 'review': False,
 'one': False,
 'thing': False
```

```
 'thing': False,
 'least': False,
 'could': False,
 'think': False,
 'may': False,
 'quit': False,
 'satisfi': False,
 'god': False,
 'want': False,
 'moneysymbnumbr': False,
 'back': False,
 'break': False,
 'usual': False,
 'daili': False,
 'find': False,
 'chicken': False,
 'biryani': False,
 'gener': False,
 'portion': False,
 'samosa': False,
 'also': False,
 'inexpens': False,
 'came': False,
 'town': False,
 'biriyani': False,
 'small': False,
 'mayb': False,
 'numbr': False,
 'piec': False,
 'naan': False,
 'pita': False,
 'bread': False,
 'heat': False,
 'hard': False,
 'paid': False,
 'curri': False,
 'huge': False,
 'amount': False,
 'rice': False,
 'plate': False,
 'much': False,
 'wast': False,
 'almost': False,
 'disappoint': False,
 'authent': False}
```

In [23]:

```
# Now lets do it for all the reviews
messages = list(zip(processed, Y))

# define a seed for reproducibility
seed = 1
np.random.seed = seed
np.random.shuffle(messages)

# call find_features function for each review
featuresets = [(find_features(text), label) for (text, label) in messages]
```

In [24]:

```
# we can split the featuresets into training and testing datasets using sklearn
from sklearn import model_selection

# split the data into training 75% and testing datasets 25%
training, testing = model_selection.train_test_split(featuresets, test_size = 0.25, random_state=se
ed)
```

In [25]:

```
print('Training:{}'.format(len(training)))
print('Testing:{}'.format(len(testing)))
```

```
Training:82
```

```
Testing:28
```

4. Scikit-Learn Classifiers with NLTK

In [26]:

```
#import all classifiers to check which one has better accuracy
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Define models to train
names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD
Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
    SVC(kernel = 'linear')
]

models = zip(names, classifiers)


#wrap models in NLTK
from nltk.classify.scikitlearn import SklearnClassifier
for name, model in models:
    nltk_model = SklearnClassifier(model)
    nltk_model.train(training)
    accuracy = nltk.classify.accuracy(nltk_model, testing)*100
    print("{} Accuracy: {}".format(name, accuracy))
```

```
K Nearest Neighbors Accuracy: 60.71428571428571
Decision Tree Accuracy: 57.14285714285714
Random Forest Accuracy: 60.71428571428571
Logistic Regression Accuracy: 53.57142857142857
SGD Classifier Accuracy: 67.85714285714286
Naive Bayes Accuracy: 57.14285714285714
SVM Linear Accuracy: 67.85714285714286
```

```
C:\Users\sagar\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The defa
ult value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\sagar\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: De
fault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [27]:

```
# Ensemble methods - Voting classifier
from sklearn.ensemble import VotingClassifier

names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD
Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
```

```
    SVC(kernel = 'linear')
]

models = list((zip(names, classifiers)))

nltk_ensemble = SklearnClassifier(VotingClassifier(estimators = models, voting = 'hard', n_jobs = -
1))
nltk_ensemble.train(training)
accuracy = nltk.classify.accuracy(nltk_model, testing)*100
print("Voting Classifier: Accuracy: {}".format(accuracy))
```

```
Voting Classifier: Accuracy: 67.85714285714286
```

In [28]:

```
# make class label prediction for testing set
txt_features, labels = list(zip(*testing))

prediction = nltk_ensemble.classify_many(txt_features)
```

In [29]:

```
# print a confusion matrix and a classification report
print(classification_report(labels, prediction))

pd.DataFrame(
    confusion_matrix(labels, prediction),
    index = [['actual', 'actual'], ['Real', 'Fake']],
    columns = [['predicted', 'predicted'], ['Real', 'Fake']])
```

```
              precision    recall  f1-score   support

           0       0.77      0.59      0.67        17
           1       0.53      0.73      0.62        11

    accuracy                           0.64        28
   macro avg       0.65      0.66      0.64        28
weighted avg       0.68      0.64      0.65        28
```

Out[29]:

		predicted	
		Real	**Fake**
actual	Real	10	7
	Fake	3	8

In [ ]:

```

```